Honey Bee Image Classification

DATS6203 Machine Learning 2, Prof: Amir Jafari
Group 4: Ruyue Zhang, Phoebe Wu, Yijia Chen, Xiaochi Ge

Introduction

This project goal is to use deep learning network to classify Bees Subspecies and Hive Health. The data is from Kaggle: https://www.kaggle.com/jenny18/honey-bee-annotated-images/kernels
In this project, we first explore the dataset and operate data preprocessing. Then we use 2 different neural network frameworks to classify Bee Subspecies and Hive Health. In this project, we use Gradient Descent Algorithm to optimize the network. In the end we evaluate the result by f1 score, AUC and ROC and test accuracy.

Description of Dataset

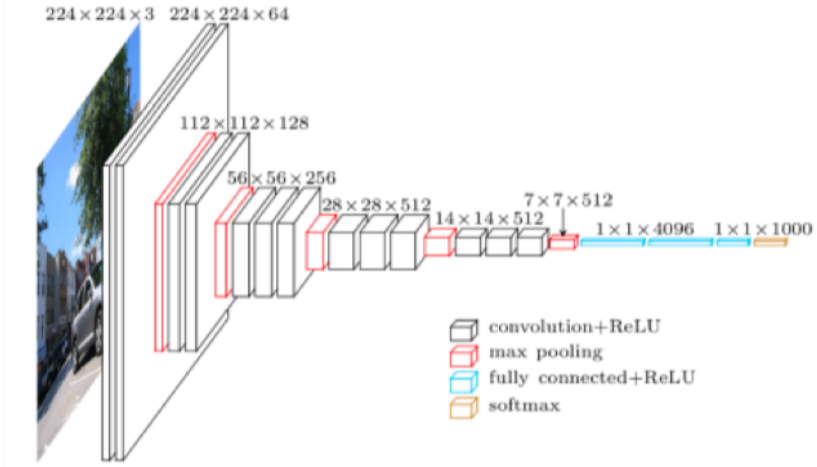
This dataset contains 5,172 bee images annotated with location, date, time, subspecies, hive health condition, caste, and pollen carrying.
You could download the data from this link: http://storage.googleapis.com/group4_data/input.zip

Deep learning network

Deep learning network is a set of learning models with complex architectures using different transfer functions and training algorithm in deep cascade of layer to get the results. Which includes Multilayer perceptron, Convolution neural network and Recurrent neural network.

- Convolution neural network

A convolution network is a multilayer feedforward network that has two- or three-dimensional inputs. CNN particularly adapted for image processing.

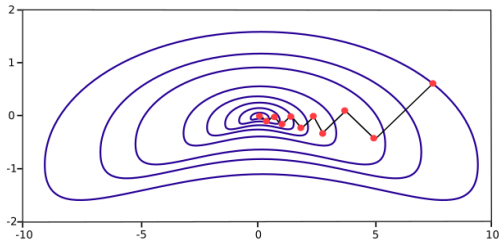


Training algorithm

Training algorithm is the procedure used to carry out the learning process in a neural network. There are many different training algorithm. The most common training algorithm are Gradient descent and Newton's method.

- Gradient descent

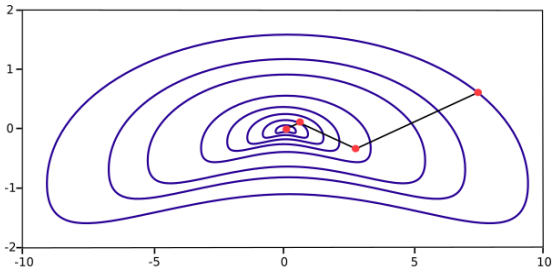
The equation: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$



- Newton's method

The objective of this method is to find better training direction by using the second derivative of the loss function.

The equation: $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1} \mathbf{g}_k$



Exploratory Data Analysis

The pie charts showed below are the percentage of geographical location,subspecies and hive health conditions of bees in the dataset; the bar charts represent the number of pollen-carrying bees and types of caste among bees in this dataset.

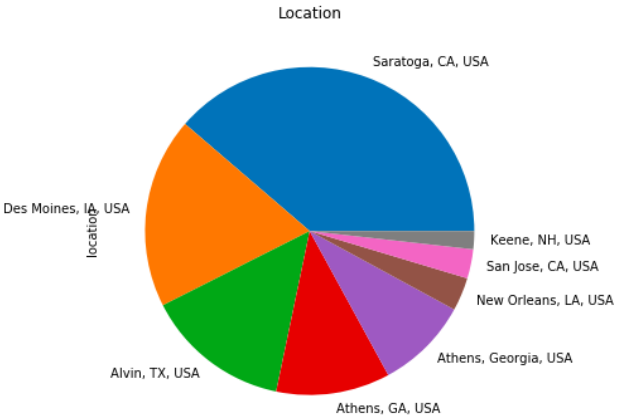


Figure 1

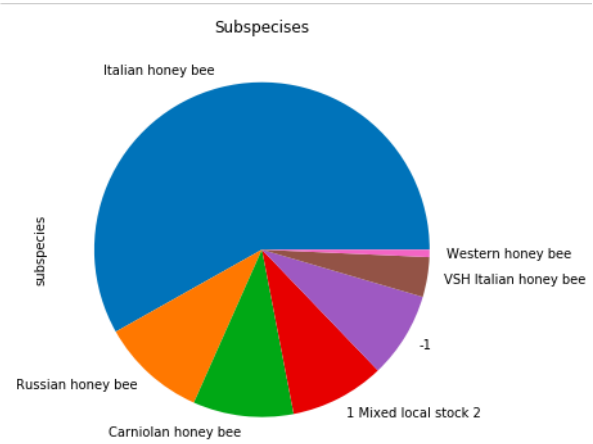


Figure 2

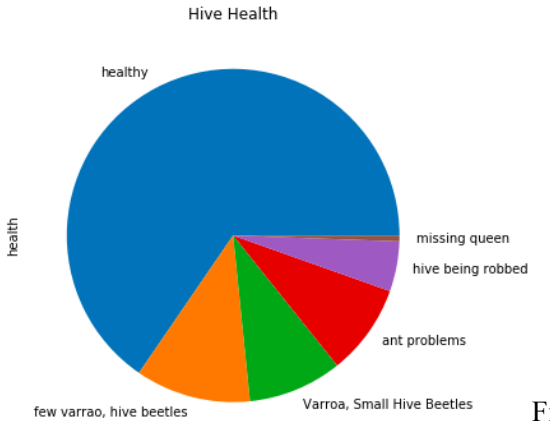


Figure3

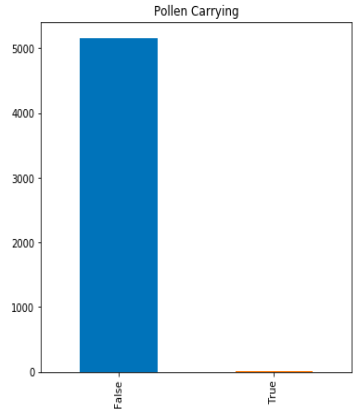


Figure 4

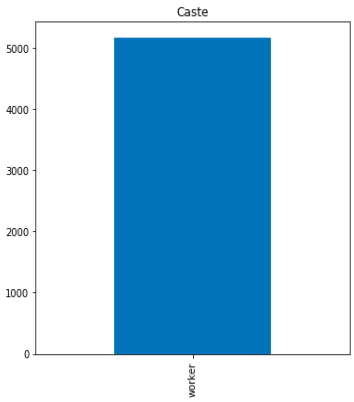


Figure 5

Data Preprocessing

- **Subspecies**

  There are in total 7 subspecies. In data preprocessing, we converted strings into numbers.

- **Hive Health**

  For Hive Health classification, as the health data is not balance (Figure 3), most of the hive are healthy. A skewed data would result in a low accuracy. So we decided to convert the rest to unhealthy. That gives us figure 4, a better place to apply classification.

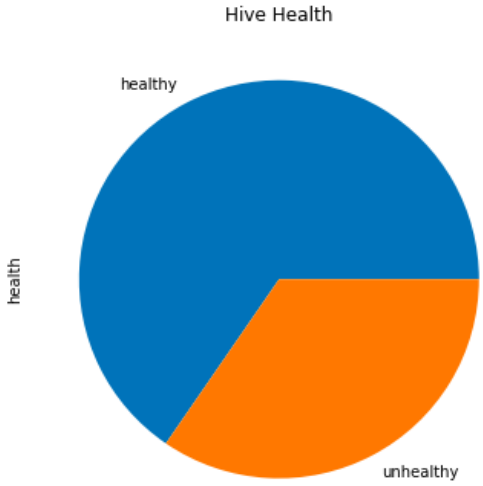


Figure 4

- **Train Test Split**

  Use sklearn to do train test split. Here we put 70% for train dataset and 30% for test dataset. Thus we have 3620 training images and 1552 testing images.

  In this project, we chose two Deep learning Frameworks: Keras and Pytorch. For each framework, we classify subspecies of the bees and the health condition of the hive from the bee images.

Keras

1. **Subspecies Classification**

   - **Data Preprocessing**

   After splitting the data. Using show_img function to get images for train set and test set. Scaled images data to [0,1] using np.stack.

```
img_wid = 120
img_len = 120
img_channels = 3

#get image
def show_img(file):
    img = skimage.io.imread(os.path.join(img_dir, file))
    img = skimage.transform.resize(img, (img_wid, img_len), mode='reflect')

    return img[:,:,:img_channels]


train_img = np.stack(x_train.apply(show_img))
print(train_img.shape)
```

```
(3620, 120, 120, 3)
```

```
test_img = np.stack(x_test.apply(show_img))
```

Encoded training label and test label using label ecode and made it one hot by keras.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(y_train)
encoded_Y = le.transform(y_train)
encoded_yt = le.transform(y_test)
```

```
nb_classes=7
```

```
from keras.utils import np_utils
y_tr = np_utils.to_categorical(encoded_Y,nb_classes)
y_te=np_utils.to_categorical(encoded_yt,nb_classes)
```

```
y_tr[0]
```

```
array([0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

- **Building Model**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 120, 120, 50)      3800
_________________________________________________________________
activation_9 (Activation)    (None, 120, 120, 50)      0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 60, 60, 50)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 60, 60, 50)        22550
_________________________________________________________________
activation_10 (Activation)   (None, 60, 60, 50)        0
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 30, 30, 50)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 45000)             0
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 315007
=================================================================
Total params: 341,357
Trainable params: 341,357
Non-trainable params: 0
_________________________________________________________________
```

First tried basic CNN model with 2 convolution layer (total 8 layers) and one fully connected layer. The activation is softmax. Then I experimented with different layers, kernel size, kernel initialization, activation, learning rate,loss function, and optimizer

(everytime change one hyperparameter). Added early stopping and tried dropped out to prevent overfitting. All max pool size 2x2.

```
stop1=EarlyStopping(monitor='loss',  patience=5, verbose=0, mode='auto')
```

Workflow

Basic cnn model ---- trying different epochs (20,30,50)---- trying activation (softmax,relu,sigmoid) ---- trying optimizers(adam, sgd) ----trying adding sample weights ---- trying adding convolutional layer (2-4) ----trying kernel size (5,3) ----trying different kernel initialization (glorot uniform,he_normal) ----trying adding other layers(dropout, batch normalization) ---- trying custom loss function for imbalanced data(weighted CE loss, Focal loss) ---- pick the best model changing the learning rate and batch size

model_name	Layers	ACC	F1	Kappa	Kernel size	Kernel initialization	Optimizer	Loss fuc	Activation	Time (s)	Epoch	Batch size
modelS	2 conv total 8	0.90	0.91	0.86	5,5,3,3	Glorot uniform	Adam	Categorical_CE	softmax	166	50	32
modelS	2 conv total 8	0.89	0.90	0.83	5,5,3,3	Glorot uniform	Adam	Categorical_CE	softmax	67	20	32
modelS	2 conv total 8	0.89	0.89	0.824	5,5,3,3	Glorot uniform	Adam	Categorical_CE	softmax	95	30	32
modelR	2 conv total 8	0.56	0.41	0.0	5,5,3,3	Glorot uniform	Adam	Categorical_CE	Relu	177	50	32
model_sg	2 conv total 8	0.97	0.91	0.86	5,5,3,3	Glorot uniform	Adam	binary_CE	sigmoid	179	50	32
modelS_S	2 conv total 8	0.77	0.72	0.059	5,5,3,3	Glorot uniform	SGD	Categorical_CE	softmax	165	50	32
modelS_sa	2 conv total 8	0.91	0.91	0.86	5,5,3,3	Glorot uniform	Adam	Categorical_CE	softmax	164	50	32
model45	4 conv total 15	0.91	0.91	0.86	5,5,3,3,3,3,2,2,	Glorot uniform	Adam	Categorical_CE	softmax	134	37	32
model43	4 conv total 15	0.92	0.93	0.88	3,3,3,3,3,3,2,2	Glorot uniform	Adam	Categorical_CE	softmax	178	50	32
model43	4 conv	0.91	0.92	0.867	3,3,3,	he_normal	Adam	Catego	softm	83	27	32

_h	total 15	6			3,3,3, 2,2,			rical_ CE	ax			
model43 _d	4 conv total 15	0.91 8	0.92	0.873	3,3,3, 3,3,3, 2,2,	Glorot uniform	Adam	Categorical_ CE	softmax		33	32
model33 _b	3 conv total 18	0.88	0.89	0.82	3,3,3, 3,2,2	Glorot uniform	Adam	Categorical_ CE	softmax	187	30	32
model43 _b	4 conv total 23	0.90	0.91	0.859	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	Categorical_ CE	softmax	263	47	232
model43 _bdr	4 conv total 23	0.92	0.93	0.88	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	Categorical_ CE	softmax	314	48	32
model43 _w	4 conv total 15	0.89	0.89	0.82	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	weighted_categorical_ CE	softmax	83	22	32
model43 _f	4 conv total 15	0.93	0.93	0.89	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	Focal loss	sigmoid	180	50	32
model43lr005	4 conv total 15	0.86	0.86	0.77	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	Categorical_ CE	softmax	185	50	32
model43lr0007	4 conv total 15	0.92	0.93	0.88	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	Categorical_ CE	softmax	178	50	32
model43lrchaing	4 conv total 15	0.74	0.70	0.54	3,3,3, 3,3,3, 2,2	Glorot uniform	SGD	Categorical_ CE	softmax	171	50	32
tmodel43 64	4 conv total 15	0.91	0.92	0.86	3,3,3, 3,3,3, 2,2	Glorot uniform	Adam	Categorical_ CE	softmax	164	48	64

Since our data is imbalanced, also tried adding weights to sample, using custom weighted cross entropy loss and Focal loss function to handle it and compared each model. Meanwhile, due to the imbalance, the accuracy can not be counted as “accurate”. F1 score is much more important.

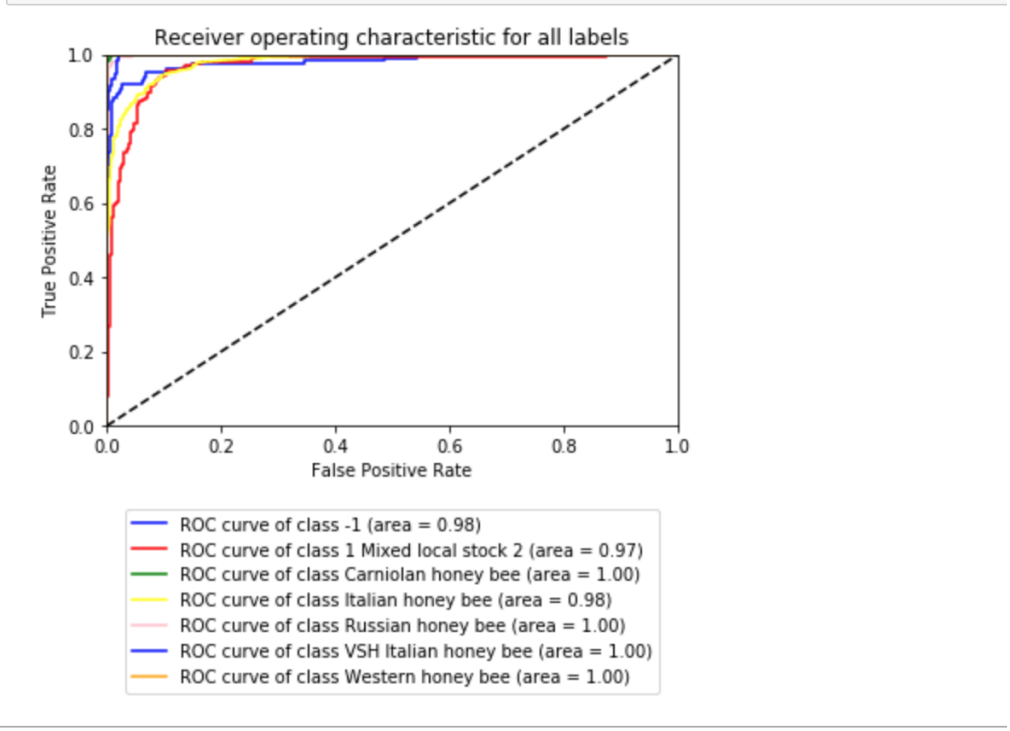
Adapting Cohen Kappa score to decide how much better the classifier is comparing with random guessing.

Conclusion:

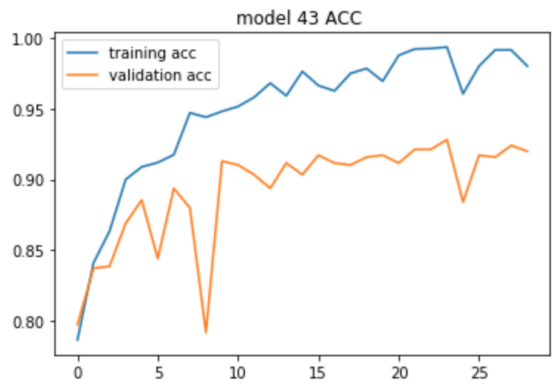
After trying, the best model are model43 and model43_f with focal loss according to f1 score, kappa score and time(model43 2s faster than model43_f). Then Compare them with ROC_AUC, model43_f did not outperform model43. Therefore focal loss seems not outperform CE in terms of traditional classification when dealing with this imbalanced data. Except for imbalance, another defect for our dataset is data size. We only have around 5000 images. The training size is only around 36000. In the future, I should try with keras image generator then use fit.generator to train the model. It can be used for image augmentation. Through keras image generator images can be flipped, shifted then act as new training samples. This way the training data could be increased.

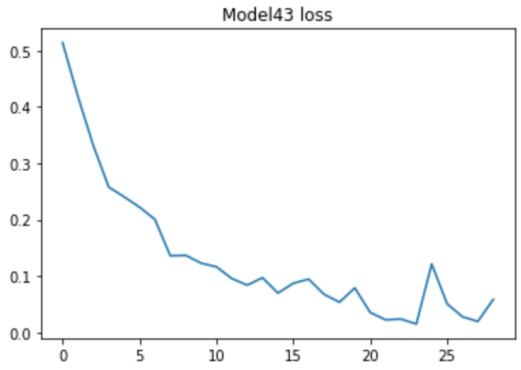
- ROC AUC Loss ACC

Model 43



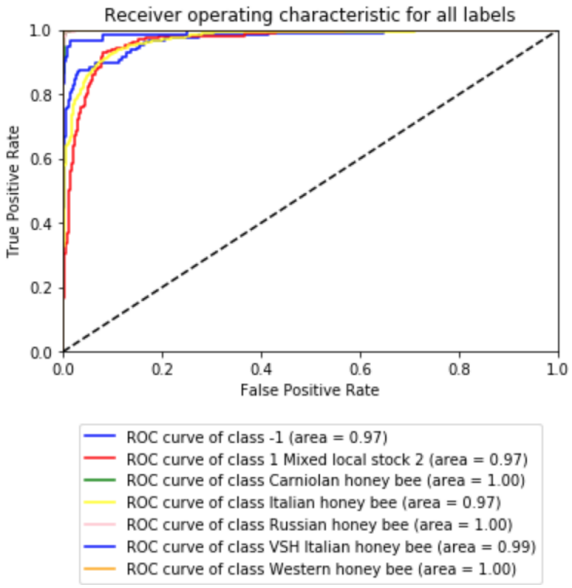
```
Text(0.5, 1.0, 'model 43 ACC ')
```



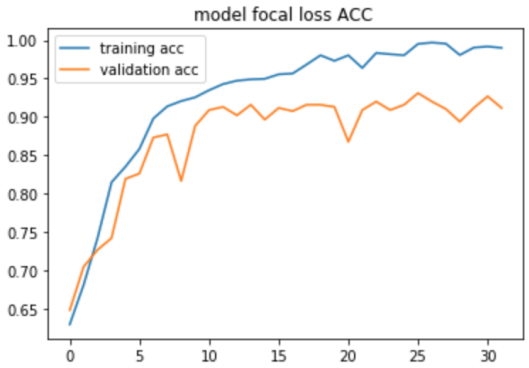


Model43_f

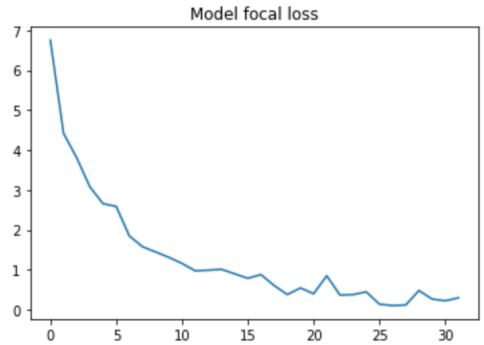
```
show_roc_auc(model43_f)
```




```
Text(0.5, 1.0, 'model focal loss ACC ')
```

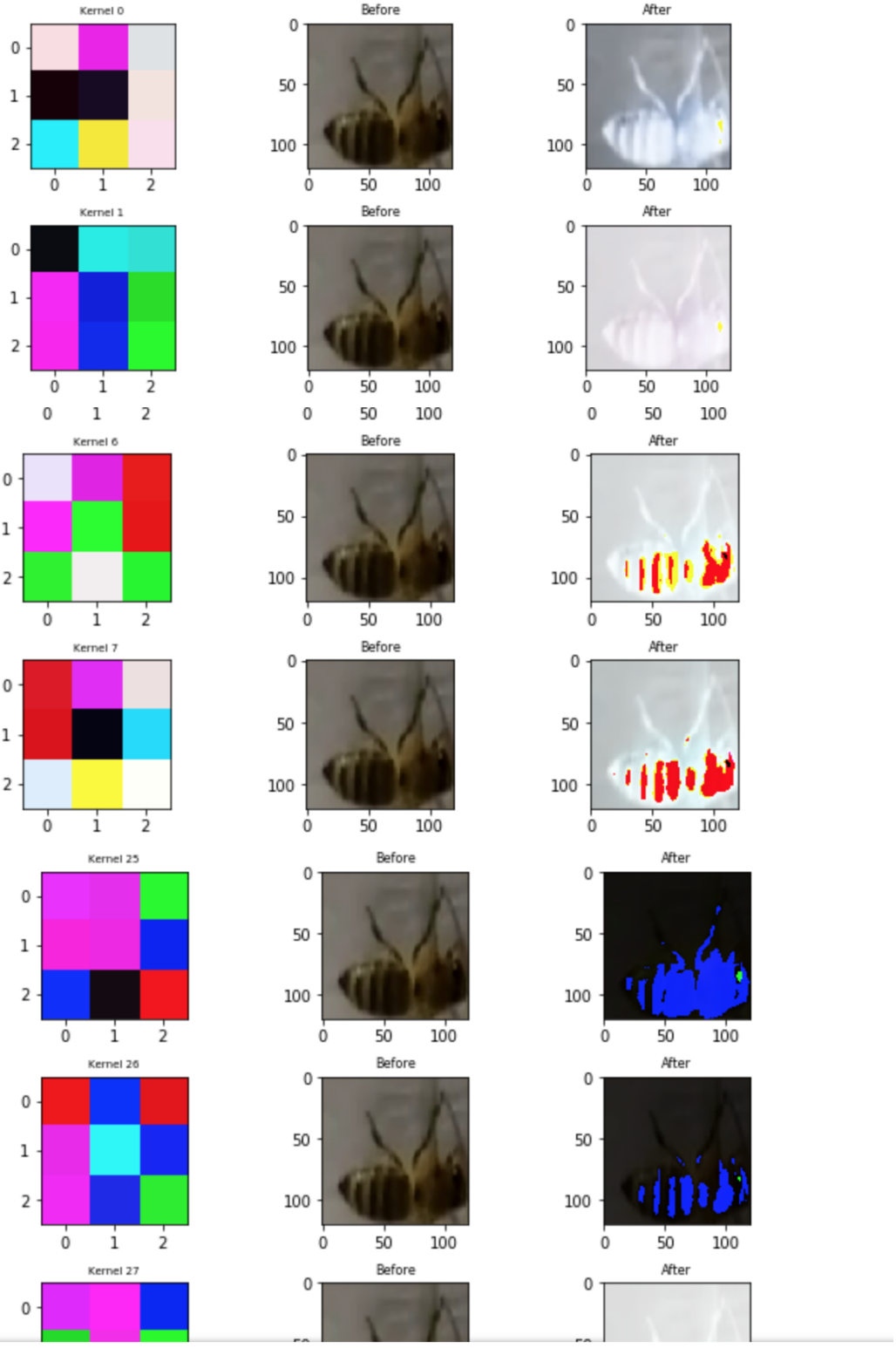


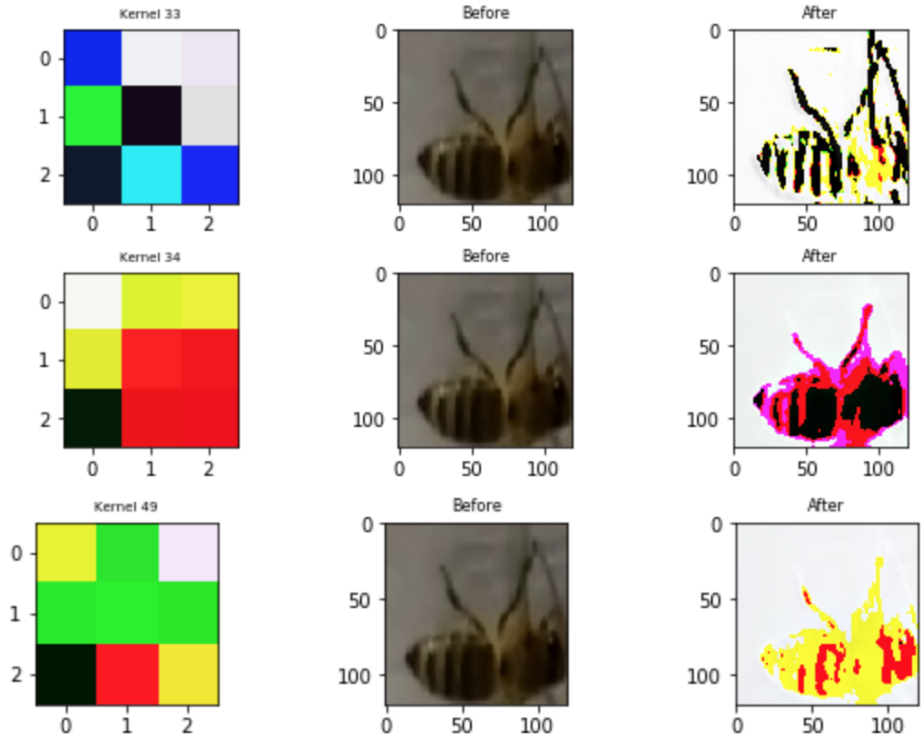

```
Text(0.5, 1.0, 'Model focal loss')
```



- Kernel visualization for one image

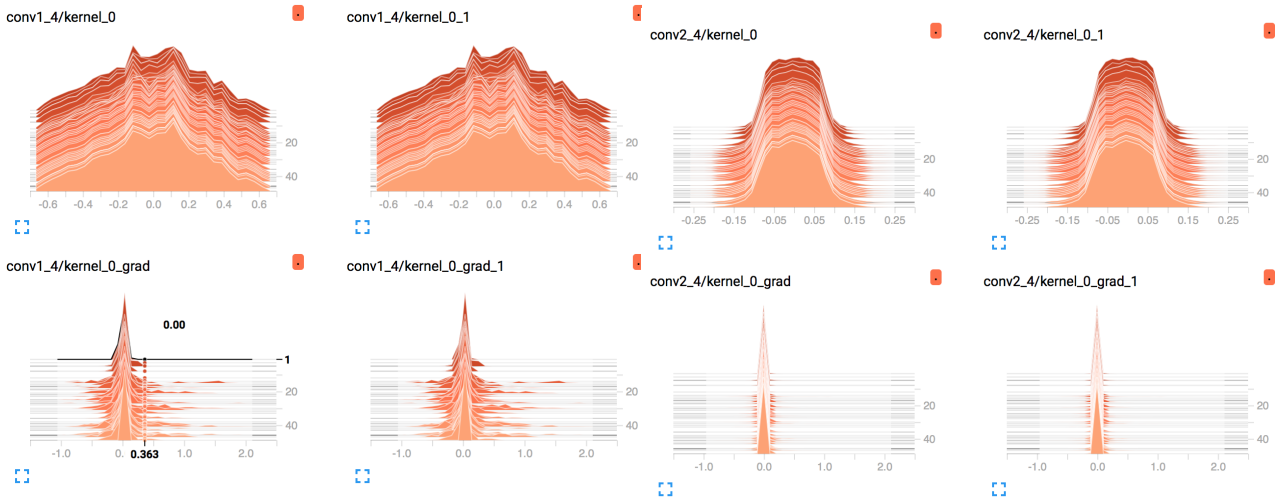
-

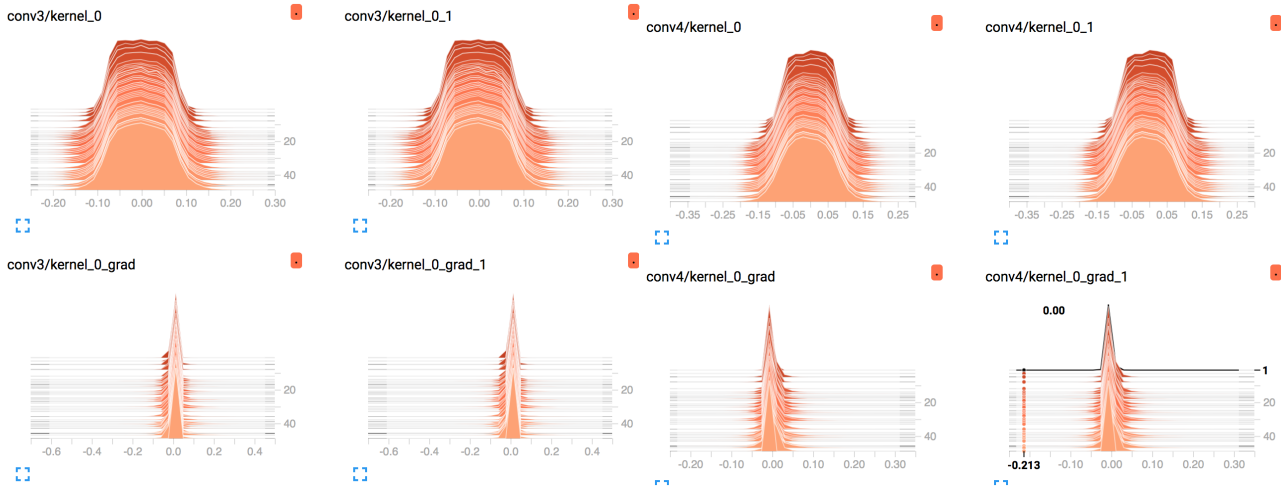

Kernel 0
Before
After
Kernel 1
Before
After
Kernel 6
Before
After
Kernel 7
Before
After
Kernel 25
Before
After
Kernel 26
Before
After
Kernel 27
Before
After



- Tensorboard for model 43

  tensorboard --logdir ./Graph





Comparing the variance, std for kernels at each convolution layer, it did get trained.

2. Hive Health Classification

- Batch Normalization and Without Normalization

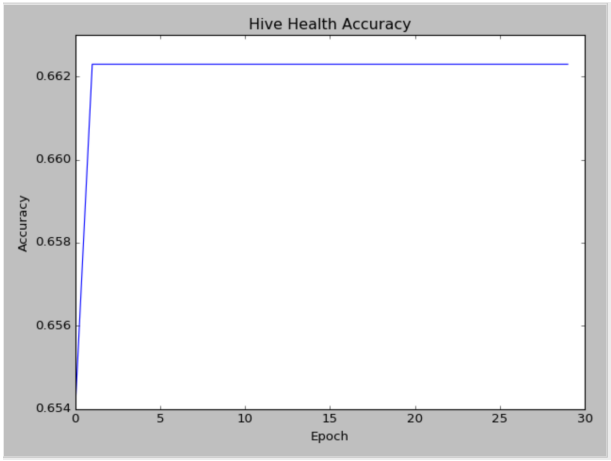
```
#model architecture
def cnn():
    model = Sequential()
    model.add(
        Conv2D(input_shape=(train_img.shape[1], train_img.shape[2], train_img.shape[3]), filters=50, kernel_size=(3, 3),
               strides=(1, 1), padding='same', kernel_initializer='he_normal'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(filters=50, kernel_size=(3, 3), strides=(1, 1), padding='same'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=50, kernel_size=(3, 3), strides=(1, 1), padding='same'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    # dense layer with 50 neurons
    model.add(Dense(50, activation='relu'))
    model.add(Dense(7, activation='softmax'))

    adam = optimizers.Adam(lr=0.001)

#fit trainning data model
training1 = model.fit(train_img, y_train, batch_size=64, validation_split=0.2, epochs=30, verbose=1)

training2 = model.fit_generator(generator.flow(train_img, y_train, batch_size=32)
                                , epochs=30, verbose=1)
#print(training2.history.keys())

#accuracy = training1.history['acc']
#loss = training1.history['loss']
```

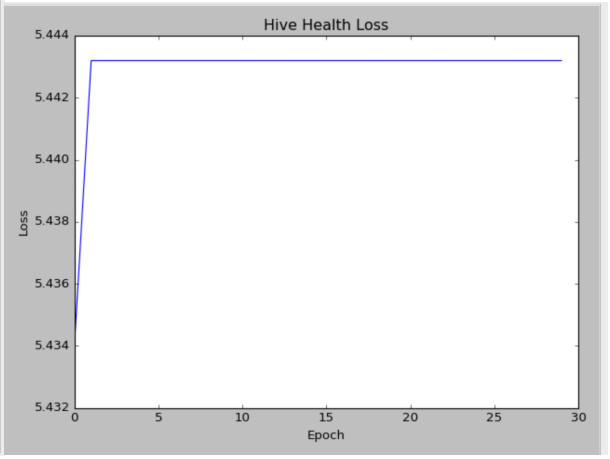
- With Batch Normalization (Epochs = 30)

	Training 1(Test Accuracy)	Training 2(Test Accuracy)

Batch Size = 32	**0.631**	**0.631**
Batch Size = 64	**0.631**	**0.631**

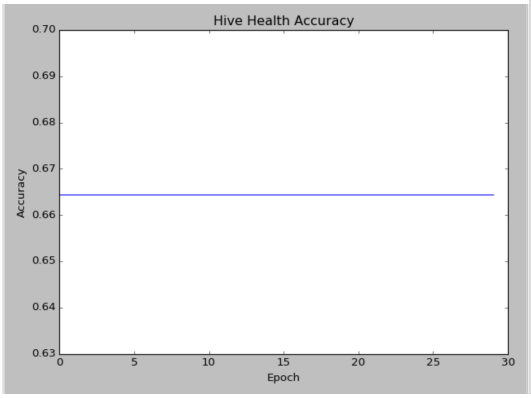
- Training 1 Accuracy and Loss (Batch size = 32, Epochs = 30)
- Test Accuracy：0.631

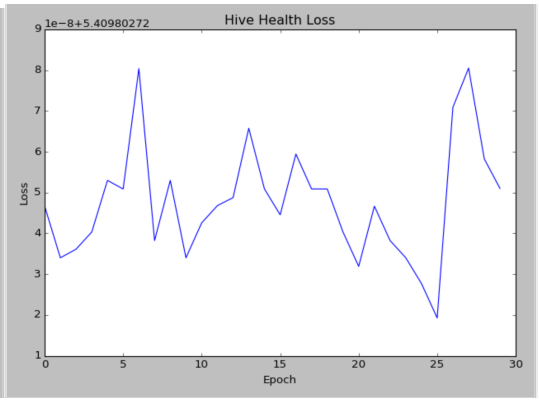




	precision	recall	f1-score	support
healthy	0.63	1.00	0.77	979
unhealthy	0.00	0.00	0.00	573
avg / total	0.40	0.63	0.49	1552

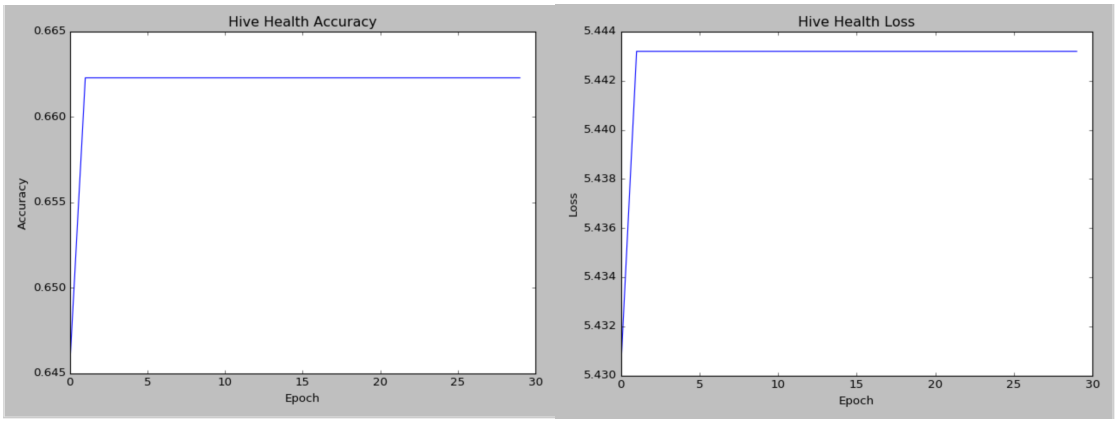
- Training 2 Accuracy and Loss (Batch size = 32, Epochs = 30)
- Test Accuracy: 0.631





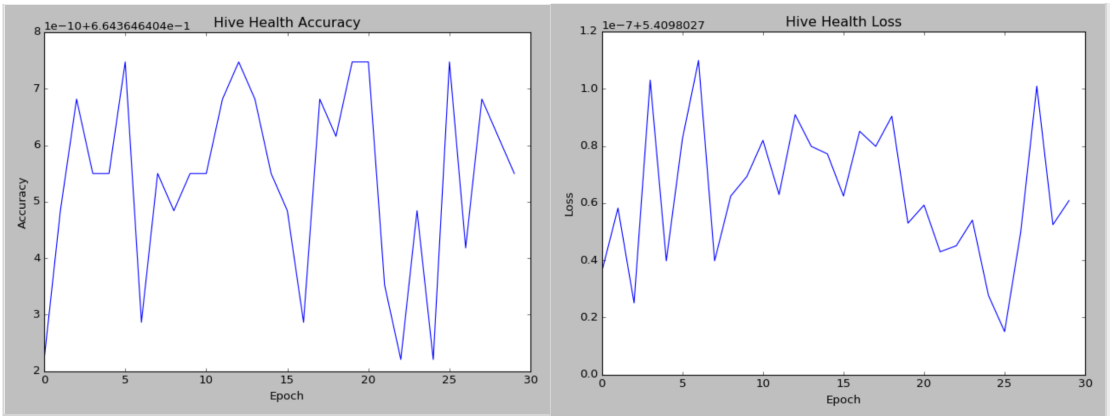
	precision	recall	f1-score	support
healthy	0.63	1.00	0.77	979
unhealthy	0.00	0.00	0.00	573
avg / total	0.40	0.63	0.49	1552

- ❖ Training 1 Accuracy and Loss (Batch size = 64, Epochs = 30)
- ❖ Test Accuracy: 0.631



	precision	recall	f1-score	support
healthy	0.63	1.00	0.77	979
unhealthy	0.00	0.00	0.00	573
avg / total	0.40	0.63	0.49	1552

- ❖ Training 2 Accuracy and Loss (Batch size = 64, Epochs = 30)
- ❖ Test Accuracy: 0.631



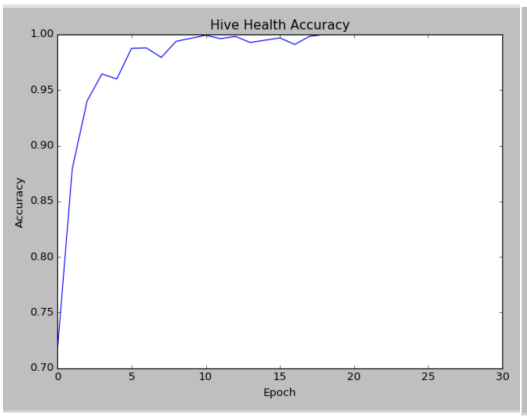
	precision	recall	f1-score	support
healthy	0.63	1.00	0.77	979
unhealthy	0.00	0.00	0.00	573
avg / total	0.40	0.63	0.49	1552

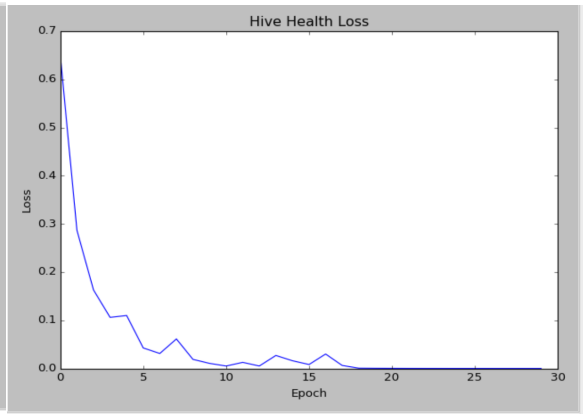
- **Without Batch Normalization (Epochs = 30)**

	Training 1(Test Accuracy)	Training 2(Test Accuracy)
Batch Size = 32	**0.995**	**0.992**

Batch Size = 64	**0.989**	**0.987**

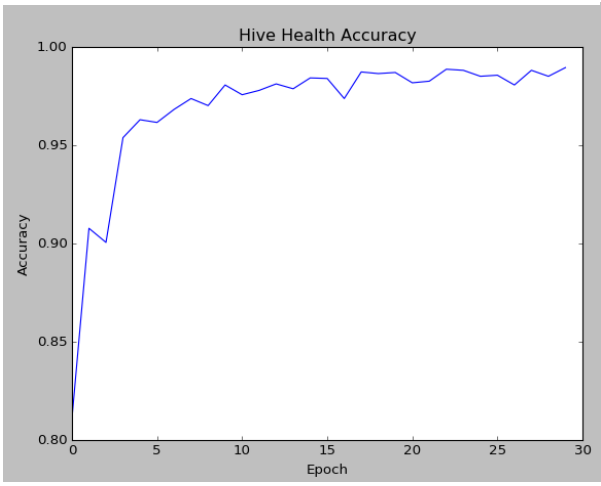
- Training 1 Accuracy and Loss (Batch size = 32, Epochs = 30)
- Test Accuracy: 0.995

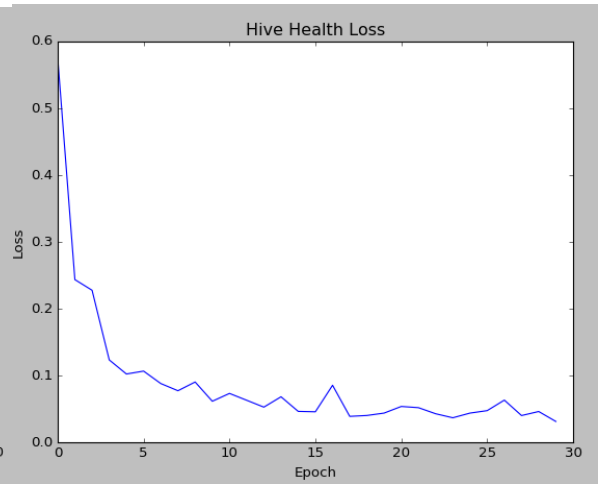




	precision	recall	f1-score	support
healthy	1.00	1.00	1.00	979
unhealthy	0.99	0.99	0.99	573
avg / total	1.00	1.00	1.00	1552

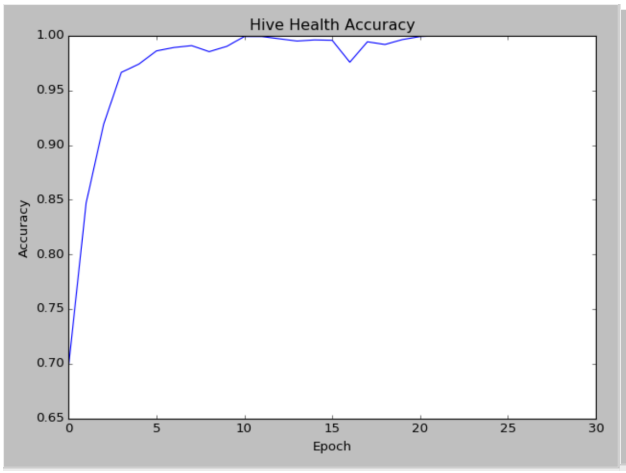
- Training 2 Accuracy and Loss (Batch size = 32, Epochs = 30)
- Test Accuracy: 0.992

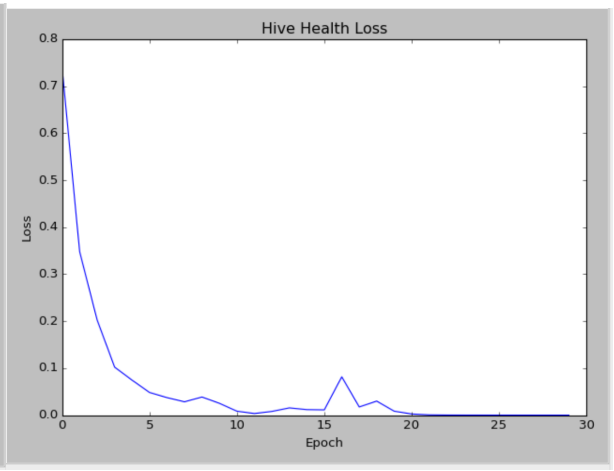




	precision	recall	f1-score	support
healthy	0.99	0.99	0.99	979
unhealthy	0.99	0.99	0.99	573
avg / total	0.99	0.99	0.99	1552

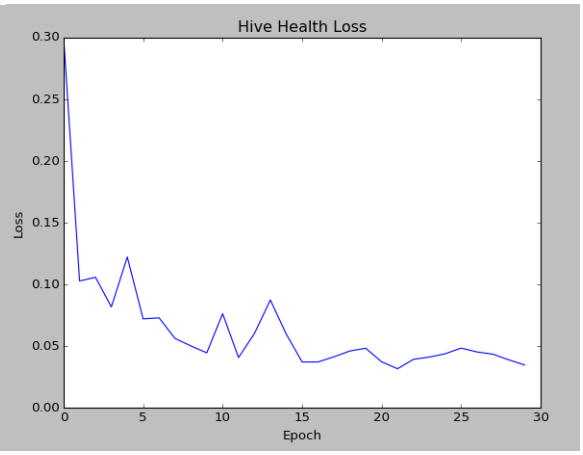
- Training 1 Accuracy and Loss (Batch size = 64, Epochs = 30)
- Test Accuracy: 0.989





	precision	recall	f1-score	support
healthy	0.99	0.99	0.99	979
unhealthy	0.99	0.98	0.99	573
avg / total	0.99	0.99	0.99	1552

- Training 2 Accuracy and Loss (Batch size = 64, Epochs = 30)
- Test Accuracy: 0.987



	precision	recall	f1-score	support
healthy	0.99	0.99	0.99	979
unhealthy	0.99	0.98	0.98	573
avg / total	0.99	0.99	0.99	1552

Conclusion:

After several experiments with normalization, various batch sizes, and different training sets, we find out that, without normalization, we can find a satisfied accuracy result. There is no much distinctions in accuracy when we change the batch size from 32 to 64.

Pytorch

1. Subspecies Classification

- Data Loading

Loading data in Pytorch is separated in 2 parts. First, the data has to be wrapped into a dataset class. Then use a Dataloader to actually read the data and put into memory. Figure 5 shows the class for this dataset. Under the class, we first use 'Image.open' to open the image from the directory, then resize all images to same size since the original images have different length and width. Also convert images to tensor since we are using pytorch. All images are colorful with RGB, we normalized it for a better training purpose. Here we also assign the label corresponding to the image for future use.

```
class honeybee(Dataset):
    def __init__(self, data, transform = None):
        self.data = data
        self.img_dir = '../input/bee_imgs'
        self.transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225])
                        ])

    def __getitem__(self, index):
        img = os.path.join(self.img_dir, self.data.iloc[index,0])
        image = Image.open(img)
        image = image.resize((120,120))
        image = image.convert('RGB')
        image = self.transform(image)
        label = np.asarray(self.data.iloc[index, 5])  # type: object

        return image,label

    def __len__(self):
        return len(self.data)
```

Figure 5

After creating the dataset class, we put the training and testing dataset into the class and use function torch.utils.data.DataLoader to read the data and put into memory.

- Build CNN Model

Since we are classifying the image data, we decided to use CNN as our model.
The most important part of the CNN model in Pytorch is calculating the output size each time after we ran the convolution neural network. Here is the formula to calculate output size:

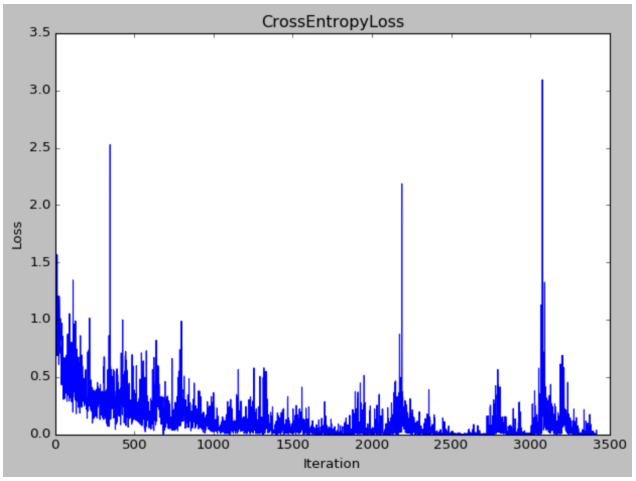
output_size = (in_size - kernel_size + 2*(padding) / stride) + 1

On each layer of the CNN model we show below, we calculated the output size and put in comment.

- **Without Dropout Node**

Epochs = 30, Batch_size = 32 - Train Loss

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride= 1, padding=2),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=2),
            nn.MaxPool2d(kernel_size=2, stride=2))
            #nn.Dropout(p=0.5)) #output size (16,61,61)

        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride= 1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2),  #output size
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=2),
            nn.ReLU()
            #nn.Dropout(p=0.5)
            ) #output size (64,34,34)

        self.layer3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
            #nn.Dropout(p=0.5))
            #output size(128,18,18)

        self.layer4 = nn.Sequential(
            nn.Linear(128*18*18, 256),
            nn.Linear(256, 128),
            nn.Linear(128, 7)
            )

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.view(out.size(0), -1)
        out = self.layer4(out)
        return out
```



Computational Time: 410.3965530395508

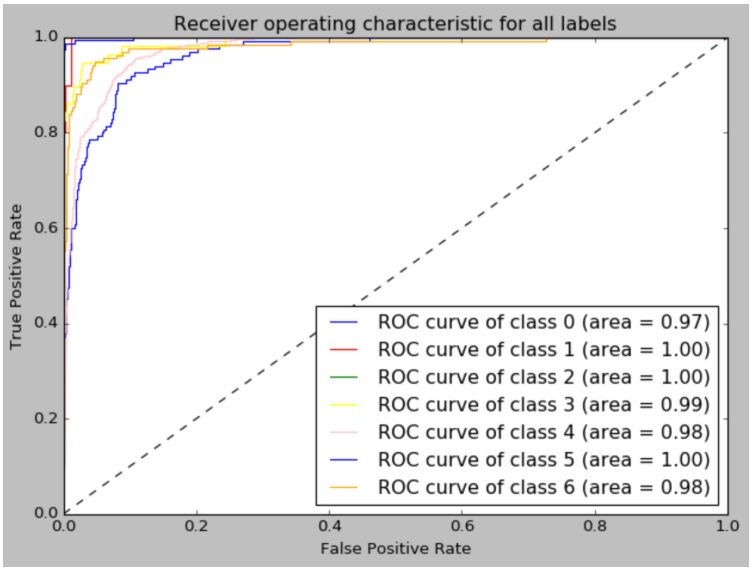
Test Accuracy of the model on the 1000 test images: 91 %

- Classification report

Classfication report:

	precision	recall	f1-score	support
1 Mixed local stock 2	0.76	0.62	0.68	135
Western honey bee	0.89	0.80	0.84	10
Carniolan honey bee	0.99	1.00	0.99	134
VSH Italian honey bee	0.98	0.78	0.87	58
Italian honey bee	0.91	0.96	0.94	914
Russian honey bee	0.98	0.99	0.98	165
-1	0.91	0.82	0.86	136
avg / total	0.91	0.92	0.91	1552

- ROC Curve

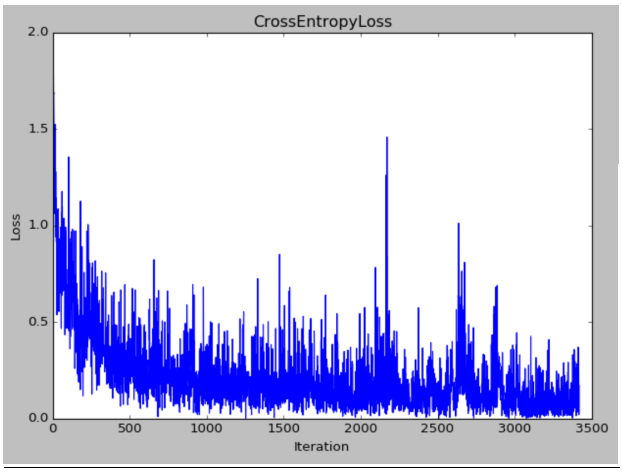


Here we see the without the dropout node, the accuracy is very high, as well as the f1-score. But we wonder if this model contains some overfitting issue so we decided to add dropout node to check.

- **With Dropout Node**

Epochs = 30, Batch size = 32

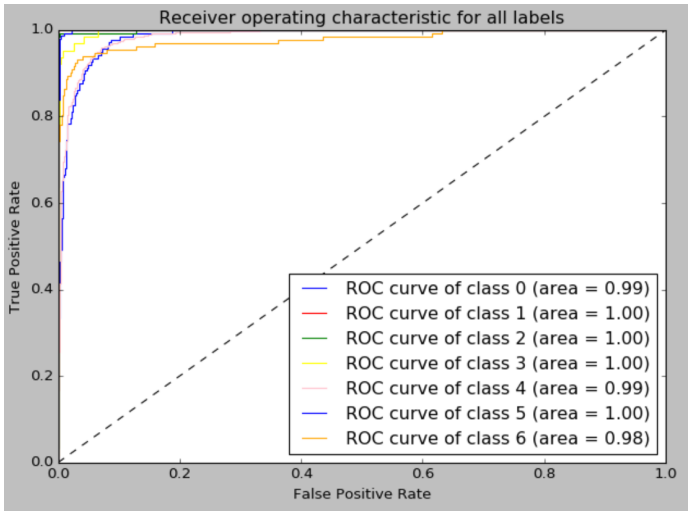
- Train Loss



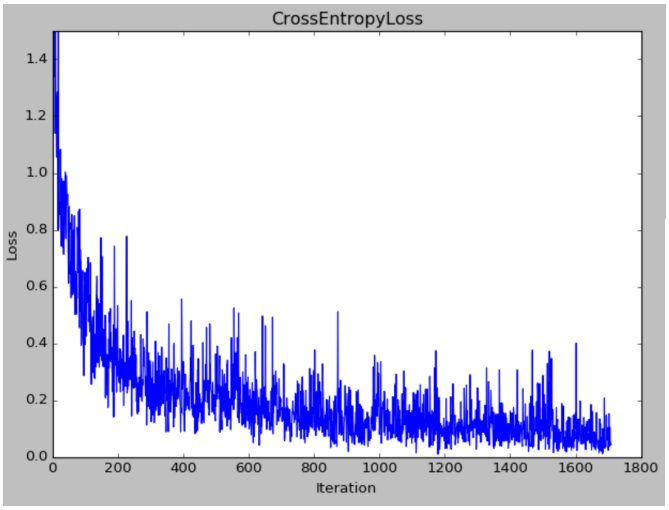
- Classification report

Classfication report:	precision	recall	f1-score	support
1 Mixed local stock 2	0.63	0.92	0.74	138
Western honey bee	0.75	1.00	0.86	6
Carniolan honey bee	0.93	0.99	0.96	142
VSH Italian honey bee	0.86	0.95	0.90	64
Italian honey bee	0.97	0.89	0.93	914
Russian honey bee	0.89	0.99	0.94	155
-1	0.97	0.76	0.85	133
avg / total	0.92	0.91	0.91	1552

- ROC Curve



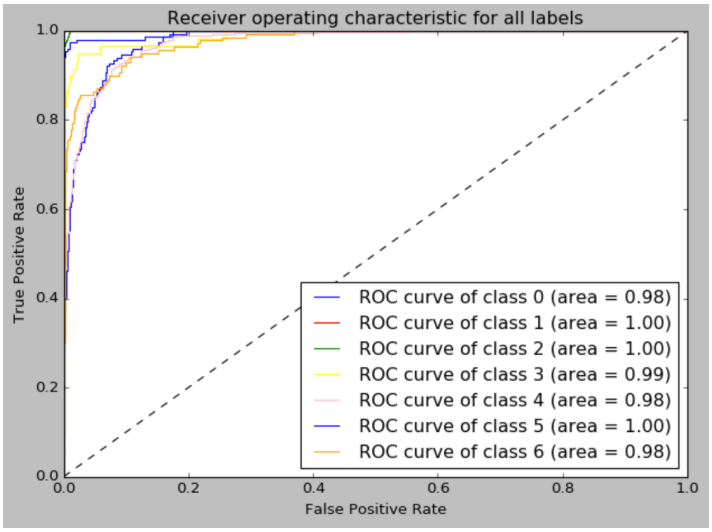
- Batch_size = 64, Epochs = 30
- Train Loss



- Classification report

```
Classfication report:
                          precision    recall  f1-score   support

1 Mixed local stock 2          0.68      0.80      0.74       152
    Western honey bee          0.44      1.00      0.61        11
  Carniolan honey bee          0.86      1.00      0.92       157
VSH Italian honey bee          0.86      0.86      0.86        59
    Italian honey bee          0.96      0.86      0.91       880
    Russian honey bee          0.82      0.98      0.89       153
                   -1          0.85      0.80      0.83       140

          avg / total          0.89      0.88      0.88      1552
```

- **ROC Curve**



- **Compare Batch size**

	Batch Size = 32	**Batch Size = 64**
Computational Time	423.86s	411.21
Test Accuracy	91%	82%
F1 Score	0.91	0.88

- **Conclusion**

Using Pytorch to run CNN for subspecies classification gives us a nice accuracy. Comparing with and without dropout node gives us only 1% difference, but adding dropout node made the f1 score of all classes over 0.7. So adding dropout node is a good choice for our model.
With dropout node, we tried using batch size 32 and batch size 64 to see if batch size would affect the model. After multiple experiments, we see the best model is with dropout node and batch size 32.

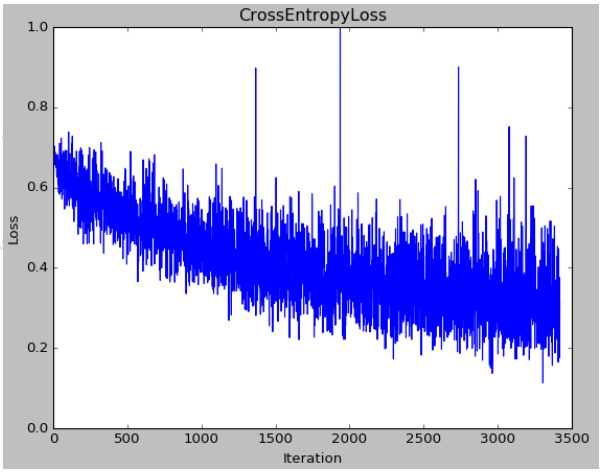
2. Hive Health Classification

- **Without Dropout node**

  Epochs = 30, Batch size = 32

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(2)) #61

        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(2)) #32

        self.layer3 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=2, stride=1), #31
            nn.MaxPool2d(kernel_size=3, stride=2) #15
            )

        self.fc1 = nn.Linear(15*15*64,64)
        self.fc2 = nn.Linear(64, 2)
```

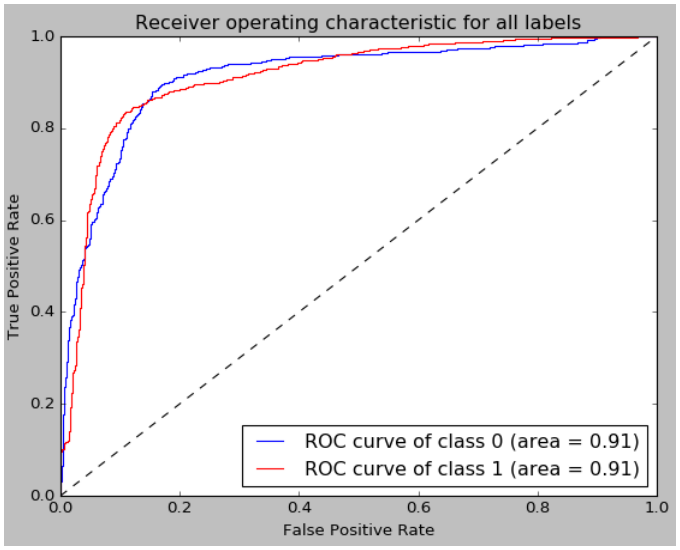
- Train loss



- Classification Report

```
('Computational Time:', 165.05251693725586)
Test Accuracy of the model on the 1552 test images: 86 %
             precision    recall  f1-score   support

    healthy       0.87      0.93      0.90      1006
  unhealthy       0.85      0.75      0.80       546

avg / total       0.87      0.87      0.86      1552
```

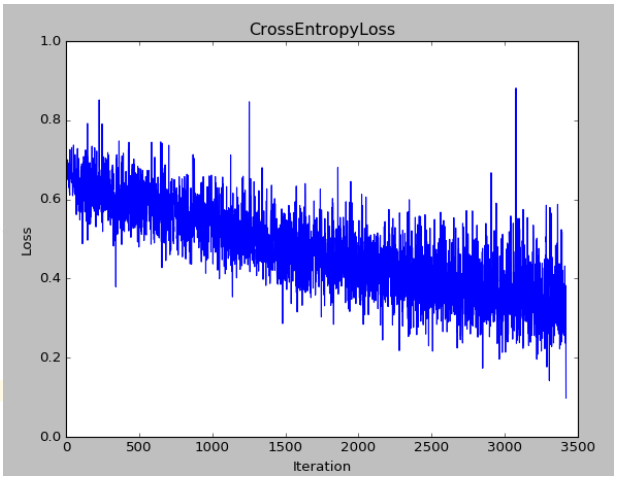
- ROC Curve



- With Dropout node:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(2)) #61

        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(2)) #32

        self.layer3 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=2, stride=1), #31
            nn.MaxPool2d(kernel_size=3, stride=2), #15
            nn.Dropout(p=0.5)
            )

        self.fc1 = nn.Linear(15*15*64,64)
        self.fc2 = nn.Linear(64, 2)
```

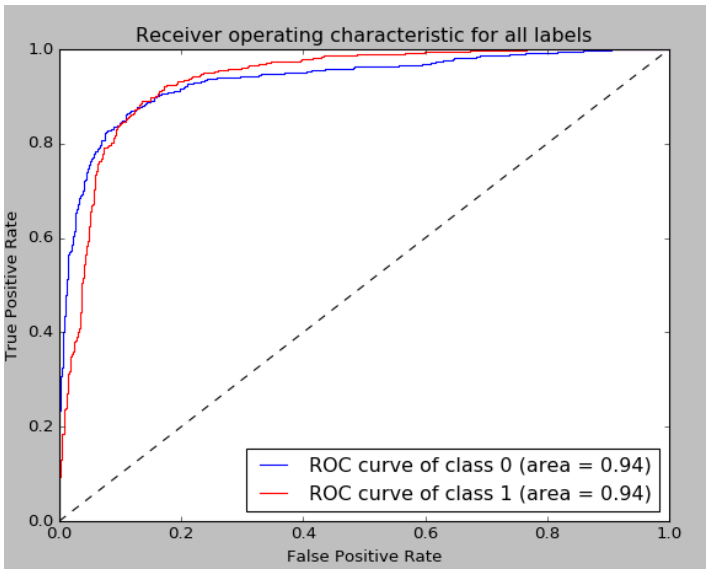
- Train loss



- Classification Report

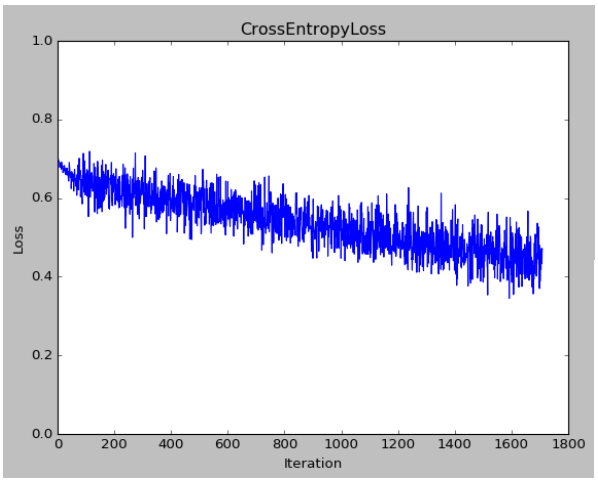
```
('Computational Time:', 198.369313955307)
Test Accuracy of the model on the 1552 test images: 87 %
             precision    recall  f1-score   support

    healthy       0.89      0.94      0.91      1035
  unhealthy       0.85      0.76      0.80       517

avg / total       0.88      0.88      0.87      1552
```

- ROC Curve



Batch_size = 64, Epochs = 30

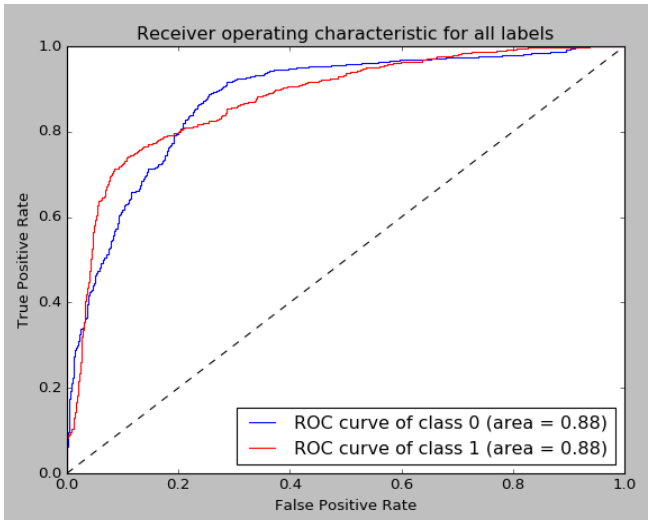
- Training loss



- Classification Report

```
('Computational Time:', 191.57918620109558)
Test Accuracy of the model on the 1552 test images: 83 %
             precision    recall  f1-score   support

    healthy       0.84      0.93      0.88      1017
  unhealthy       0.83      0.67      0.74       535

avg / total       0.84      0.84      0.83      1552
```

- ROC Curve



- Compare Batch size

	Batch Size = 32	**Batch Size = 64**
Computational Time	198.37	191.58
Test Accuracy	87%	83%
F1 Score	0.87	0.83

- **Conclusion**

From the performance, the accuracy and auc score for the model without dropout nodes are 86% and 91% respectively. On the other hand, the accuracy and auc score with dropout

nodes are 87% and 94%. However, the model with dropout node cost much computational time. Therefore, the dropout node in the mode we used to classify the hive health from the bee images do not affect a lot.

For the batch size, in our model, the small batch size(batch size = 32) shows the better overall performance than the bigger batch size(batch size = 64). Nevertheless, there is not much different in the computational time for the two batch size. Thus, the batch size does affect the performance of our model.

Summary

To sum up, for Hive Health Classification, Keras produced a better performance than Pytorch. For Subspecies Classification , Keras and Pytorch have similar performance. Compared with batch size = 64, it could yield a greater accuracy when batch size = 32. Since with smaller batch size, gradients updated more frequently.

From the project, we enhance our capability of deep learning model application skills. We tried different models and algorithms to get a better understanding of deep learning. For future research, we could use more layers and change pooling methods. If there are more images in the dataset to fix the imbalanced problem, we would have a better accuracy on some of the bee classes.

References

Jha, Shekhar. “VGG16-Honey Bee Health Classification.” *RSNA Pneumonia Detection Challenge | Kaggle*, 2018, www.kaggle.com/xanthate/vgg16-honey-bee-health-classification.

Gage, Justin. “Convolutional Neural Nets in Pytorch.” *Algorithmia Blog*, Algorithmia, 10 Apr. 2018, blog.algorithmia.com/convolutional-neural-nets-in-pytorch/.

“Torch.nn.” *PyTorch*, pytorch.org/docs/stable/nn.html.

“Data Loading and Processing Tutorial.” *PyTorch*, pytorch.org/tutorials/beginner/data_loading_tutorial.html.

Pukhov, Dmitry. “Honey Bee Health Detection with CNN.” *RSNA Pneumonia Detection Challenge | Kaggle*, www.kaggle.com/dmitrypukhov/honey-bee-health-detection-with-cnn.

_Neo__. “Pytorch--Tensor.size().” 28 Nov. 2017, blog.csdn.net/a132582/article/details/78658155.

Leigh. “PyTorch Tutorial: Dataset. Data Preparetion Stage.” *RSNA Pneumonia Detection Challenge | Kaggle*, www.kaggle.com/leighplt/pytorch-tutorial-dataset-data-preparetion-stage.

Santos, Leonardo Araujo dos. “DataLoader and DataSets.” *Deep Q Learning · Artificial Inteligence*, leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pytorch/dataloader-and-datasets.html.

Preda, Gabriel. “Honey Bee Subspecies Classification.” *RSNA Pneumonia Detection Challenge | Kaggle*, 2018, www.kaggle.com/gpreda/honey-bee-subspecies-classification.

“Image Preprocessing.” *Keras Documentation*, keras.io/preprocessing/image/.

“Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python.” *EliteDataScience*, 8 Feb. 2018, elitedatascience.com/keras-tutorial-deep-learning-in-python.

“Model Class API.” *Keras Documentation*, keras.io/models/model/.

“Sklearn.metrics.f1_score.” *1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation*, scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

"St-m-Hdstat-Rnn-Deep-Learning."Wikistat, www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-hdstat-rnn-deep-learning.pdf.

Artelnics, Alberto Quesada. "5 Algorithms to Train a Neural Network." Health Care Applications Neural Designer, Neural Designer, www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network .

Mkocabas, Muhammed. "Mkocabas/Focal-Loss-Keras." *GitHub*, 18 Mar. 2018, github.com/mkocabas/focal-loss-keras.

Shaoanlu. "Experiment: Applying Focal Loss on Cats-vs-Dogs Classification Task." *SALu*, 9 Nov. 2018, shaoanlu.wordpress.com/2017/08/16/applying-focal-loss-on-cats-vs-dogs-classification-task/.

Shaikh, Faizan, and Faizan. "Essentials of Deep Learning: Visualizing Convolutional Neural Networks in Python." *Analytics Vidhya*, 22 Mar. 2018, www.analyticsvidhya.com/blog/2018/03/essentials-of-deep-learning-visualizing-convolutional-neural-networks/.

Brownlee, Jason. "8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset." *Machine Learning Mastery*, 7 June 2016, machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

Pukhov, Dmitry. "Honey Bee Health Detection with CNN." *RSNA Pneumonia Detection Challenge | Kaggle*, 2018, www.kaggle.com/dmitrypukhov/honey-bee-health-detection-with-cnn.

BigMoyan. "初始化方法." *初始化方法 - Keras中文文档*, keras-cn.readthedocs.io/en/latest/other/initializations/.

Godoy, Daniel. "Hyper-Parameters in Action! Part II - Weight Initializers." *Towards Data Science*, Towards Data Science, 18 June 2018, towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404.

Dahal, Paras. "Classification and Loss Evaluation - Softmax and Cross Entropy Loss." *DeepNotes*, 28 May 2017, deepnotes.io/softmax-crossentropy.

"Using sample_weight in Keras for Sequence Labelling." *Stack Overflow*, 2018, stackoverflow.com/questions/48315094/using-sample-weight-in-keras-for-sequence-labelling.

Benshetler, Jeff . “Why does the accuracy decreases after I did data normalization in CNN model?” 26 Aug 2016, https://www.quora.com/Why-does-the-accuracy-decreases-after-I-did-data-normalization-in-CNN-model

Jafari, Amir. “Amir-Jafari/Deep-Learning.” *GitHub,* 15 Nov. 2018, https://github.com/amir-jafari/Deep-Learning.